View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Digital library of Brno University of Technology

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ALGORITMY VYHLEDÁVÁNÍ

SEARCHING ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ĎURIŠ
AUTHOR

VEDOUCÍ PRÁCE — RNDr. JIŘÍ DVOŘÁK, CSc.
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/2013

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Martin Ďuriš

který/která studuje v **bakalářském studijním programu**

obor: **Strojní inženýrství (2301R016)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Algoritmy vyhledávání

v anglickém jazyce:

Searching algorithms

Stručná charakteristika problematiky úkolu:

Vyhledávání dat podle zadaných hledisek patří mezi důležité problémy informatiky. Efektivnost řešení mnoha problémů řešených na počítači závisí na tom, jak efektivně je řešeno ukládání a vyhledávání dat.

Cíle bakalářské práce:

1. Charakterizovat problematiku vyhledávání se zaměřením na vyhledávání zadaného prvku v dané množině prvků.
2. Popsat vybrané principy a algoritmy vyhledávání.
3. Na základě údajů z literatury a vlastních experimentů provést jejich srovnání.

Seznam odborné literatury:

WIEDERMANN, J. Vyhledávání. Praha, SNTL 1991.
WIRTH, N. Algoritmy a štruktúry údajov. Bratislava, ALFA 1988.
HONZÍK, J. a kol. Programovací techniky. Skripta. VUT v Brně, 1985.

Vedoucí bakalářské práce: RNDr. Jiří Dvořák, CSc.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2012/13.

V Brně, dne 20.11.2012

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
DĚKANÁT

Ing. Jan Roupec, Ph.D.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Děkan

Abstrakt

Práca sa venuje opisu a analýze vybraných algoritmov vyhľadávania v jednodnorozmerných poliach. Algoritmy sú analyzované matematicky a graficky. Matematická analýza je realizovaná na základe množstva vykonaných porovnaní a pomocou literatúry. Grafická analýza vychádza z výsledkov viacerých pokusov. Každý z vybraných algoritmov bol otestovaný na náhodne vygenerovanom poli. Z výsledných časov úspešného a neúspešného vyhľadávania boli vytvorené grafy, ktoré boli použité na porovnanie jednotlivých algoritmov. Z výsledkov vyplýva, že najrýchlejší z vybraných algoritmov je algoritmus binárneho vyhľadávania.

Abstract

The thesis describes and analyzes selected searching algorithms for single dimensional array. Algorithms are analyzed mathematically and graphically. Mathematical analysis is based on analysis of average amount of performed comparisons required for the final result and literature review. Graphical analysis is based on multiple examinations of selected searching methods with randomly generated single dimensional arrays. The graphs show dependency of elapsed searching time of successful and unsuccessful searching trial on the size of single dimensional array. The results establish the binary searching algorithm as the fastest one of the selected searching algorithms.

Kľúčové slová

Algoritmus, vyhľadávanie, pole, zoznam, strom, hashovanie, lineárne, sekvenčné, binárne, interpolačné, analýza

Keywords

Algorithm, search, array, list, tree, hashing, linear, sequential, binary, interpolation, analysis

Prehlásenie o originalite

Prehlasujem, že som svoju bakalársku prácu na tému „Algoritmy vyhledávání“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia § 152 trestného zákona č. 140/1961 Sb.

Bibliografická citácia

ĎURIŠ, M. *Algoritmy vyhledávání.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 38 s. Vedoucí bakalářské práce RNDr. Jiří Dvořák, CSc..

Obsah

1 Úvod

Zber informácií, ich efektívne ukladanie a vyhľadávanie sú úlohy s ktorými je ľudstvo konfrontované už od pradávna.V súčasnosti sa tieto úlohy dostávajú do popredia hlavne z dôvodov spojených s obrovským množstvom informácií generovaných pri takmer každej ľudskej činnosti. Pokrok v technologiách zberu a ukladania dát každým dňom znásobuje objem dostupných dát až do tej miery, že hovoríme o informačnej explózi. Množstvo informácií, ktoré nám sprístupnili telekomunikačné a počítačové technológie je také obrovské v porovnaní s minulosťou, že najväčším problémom súčasnosti sa stáva úloha nájsť v uložených dátach tú správnu, či potrebnú informáciu. V súlade s tým sa vyvíjali a stále aj vyvíjajú zodpovedajúce technológie triedenia a vyhľadávania informácií od jednoduchých vyhľadávacích algoritmov až po najnovšie metódy vyhľadávania pomocou hashových tabuliek.

Jednou z najrozšírenejších dátových štruktúr je jednorozmerné pole, ktoré obsahuje dáta len jedného typu. Napr. akékoľvek technologické dáta je možné uložiť v tejto jednoduchej dátovej štruktúre. Tým však rastie význam efektívneho vyhľadávania v takýchto dátach. Táto práca je preto zameraná na analýzu vybraných algoritmov vyhľadávania v jednorozmerných poliach, ich popis a porovnanie rýchlosti s akou tieto vybrané algoritmy dokážu nájsť hľadaný prvok v poli, alebo zistiť, že prvok sa v poli nenachádza. Porovnanie jednotlivých algoritmov je realizované matematicky na základe priemerného počtu porovnaní a pokusne na náhodne vygenerovaných poliach.

2 Problematika vyhľadávania

Ľudstvo od počiatku svojej existencie sa snaží informácie zbierať a uchovávať. Jedny z najstarších zachovaných záznamov informácií sú napr. nápisy na pyramídach, či obrazy v jaskyniach, alebo literárne diela najstarších autorov z čias antiky. Snaha o sprístupnenie zaznamenaných informácií viedla od kamenných tabuliek či vytesaných nápisov na architektonických dielach cez opisovanie údajov na papyrusoch a ručne písaných kníh až po v súčasnosti najrozšírenejší spôsob zdieľania informácií v digitálnej forme na Internete. So zberom a distribúciou informácií úzko súvisela a stále súvisí problematika vyhľadávania informácií. Jednoduché metódy katalogizácie informácií sú v súčasnosti nahrádzané sofistikovanými metódami vyhľadávania za pomoci výkonných počítačových systémov. Adekvátne k druhu činnosti a druhu zbieraných informácií boli vyvinuté príslušné dátové štruktúry a samozrejme aj algoritmy ako v týchto dátových štruktúrach vyhľadávať.

2.1 Jednorozmerné polia

Jednorozmerné pole je jednou z najjednoduchších a najzákladnejších dátových štruktúr. Skladá sa z pevného počtu prvkov rovnakého typu, pričom každý prvok možno presne identifikovať podľa indexu. Každý index môže byť len ordinálneho typu a označuje práve jeden prvok. Vyhľadávanie v poli je realizované tak, že v cykle sa pomocou indexov pristupuje k jednotlivým prvkom, ktoré sú porovnávané s hľadaným prvkom. Výhodu, že sa dá pristupovať ku každému prvku priamo pomocou indexu, sa dá využiť v usporiadanom poli (tab. 2.1). Na usporiadané pole sa dajú aplikovať rôzne algoritmy, ktoré nemusia porovnávať každý prvok, ale len nejakú časť na nájdenie hľadaného prvku. Nevýhodou je udržiavanie usporiadanosti poľa. Ak máme miliónprvkové usporiadané pole a chceme do stredu vložiť nový prvok, tak musíme všetky prvky napravo od vkladaného posunúť o jednu pozíciu. Bude to päťstotisíc posunutí, čo je dosť časovo náročná operácia. [3, 12]

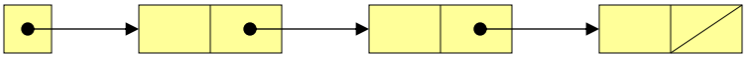
Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	2	3	5	8	11	12	15	20	21

Tabuľka 2.1: Usporiadané pole prvkov

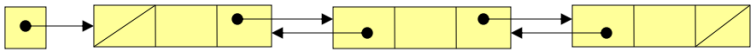
2.2 Zreťazené zoznamy

Zreťazený zoznam je údajový typ, ktorý sa skladá z jednotlivých záznamov. Tie obsahujú hodnotu prvku a smerník na nasledujúci záznam. Ak obsahujú iba jeden smerník a to na nasledujúci záznam, potom sa zoznamy skladajúce sa z takýchto záznamov nazývajú jednosmerne zreťazené (obr. 2.1). Ak obsahujú aj smerník na predchádzajúci záznam, nazývajú sa zoznamy obojsmerne zreťazené (obr. 2.2). Algoritmy pre zoznamy sú podobné ako algoritmy pre polia. Ale pri prehľadávaní zoznamu sa nedá pristupovať k záznamom priamo pomocou indexu ako v poli, ale k nasledujúcemu záznamu alebo

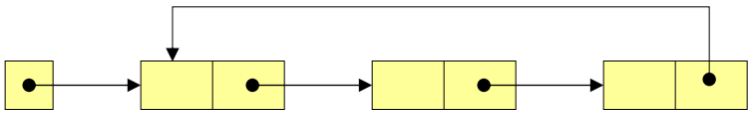
záznamu niekde uprostred zoznamu sa dá dostať len pomocou smerníka z predchádzajúceho záznamu. Niekedy môže posledný záznam obsahovať smerník na prvý záznam. Pri obojsmerne zreťazenom zozname potom aj prvý záznam môže obsahovať smerník na posledný. Takéto zoznamy sú potom nazývané zoznamy zreťazené dokola (obr. 2.3). Výhodou zreťazeného zoznamu je jednoduché vkladanie prvku. Pri jednosmernom zozname stačí zmeniť jeden smerník prípadne pri obojsmerne zreťazenom zozname dva smerníky. [12]



Obr. 2.1: Jednosmerne zreťazený zoznam



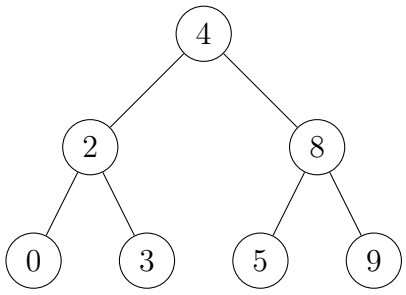
Obr. 2.2: Obojsmerne zreťazený zoznam



Obr. 2.3: Zoznam zreťazený dokola

2.3 Binárne stromy

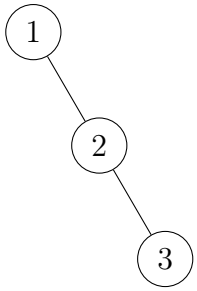
Binárny strom (obr. 2.4) je dátová štruktúra podobná stromu, v ktorej každý uzol má maximálne dvoch potomkov a každý potomok uzla môže byť rozdelený na pravý alebo ľavý. [8] V ľavom podstrome ľubovoľného uzla sa nachádzajú iba prvky menšie ako v uzle a v pravom podstrome sa nachádzajú iba prvky väčšie ako v uzle. Každý podstrom je potom samostatným binárnym stromom. [5]



Obr. 2.4: Binárny strom

Výhodou binárneho stromu je podobne ako u zreťazených zoznamov jednoduché pridávanie prvkov a veľmi rýchle vyhľadávanie. Problém nastáva len pri vkladaní dát.

Ak sú nevhodné dáta vkladané, môže vzniknúť nevyvážený strom, kde vyhľadávanie trvá rovnako dlho ako pri vyhľadávaní v zreťazenom zozname. Na obrázku 2.5 je zobrazený nevyvážený binárny strom. Ak ukladáme len celé čísla, nie je možné medzi jednotlivé uzly už nič vložiť a dostávame štruktúru podobnú zreťazenému zoznamu. [5, 16, 17, 12]



Obr. 2.5: Nevyvážený binárny strom

2.4 Hashovanie

Vyhľadávacie algoritmy polí sa dajú urýchliť ak poznáme nejaké vlastnosti uložených dát. Napríklad keď je treba vyhľadávať prvok v usporiadanom poli, nie je nutné ho prehľadávat postupne prvok po prvku, ale je možné aplikovať binárny vyhľadávací algoritmus. U hashovania je snaha vytvoriť dátovú štruktúru tak, aby sa dal akýkoľvek prvok nájsť jedným porovnaním.

Hashová tabuľka je štruktúra podobná jednorozmernému poľu. Dáta sú do nej ukladané takým spôsobom, aby ich bolo ľahké nájsť neskôr. Jednotlivé pozície v tabuľke môžu byť označné číslami 1, 2, 3 atď. Na začiatku je tabuľka prázdna. Jednoduchá hashovacia tabuľka s desiatimi pozíciami je zobrazená v tabuľke 2.2. Dáta sa do nej neukladajú za sebou ako v jednorozmernom poli ale ich pozícia sa vypočítava hashovacou funkciou.

1	2	3	4	5	6	7	8	9	10
90	2		5	16				7	

Tabuľka 2.2: Hashová tabuľka

V hashovacej tabuľke sa vyhľadáva rovnakým spôsobom, akým sa do nej ukladajú dáta. Pomocou hashovacej funkcie sa vypočíta pozícia prvku. Potom je hľadaný prvok porovnaný s prvkom na vypočítanej pozícii. V prípade zhody je prvok v tabuľke nájdený a v prípade, že je pozícia prázdna, prvok sa v tabuľke s istotou nenachádza. [18]

3 Algoritmy vyhľadávania

3.1 Algoritmy pre neusporiadané polia

Tieto algoritmy musia postupne hľadaný prvok porovnávať s každým prvkom poľa dovtedy, kým hľadaný prvok nenájdu alebo kým nevyčerpajú všetky prvky poľa. Typickým predstaviteľom tejto skupiny je algoritmus sekvenčného vyhľadávania alebo algoritmy, ktoré majú za úlohu zistiť určitú vlastnosť jedného alebo viacerých prvkov v poli.

3.1.1 Sekvenčné vyhľadávanie

Algoritmus sekvenčného vyhľadávania je základom všetkých ostatných vyhľadávacích algoritmov. Prehľadáva všetky prvky poľa postupne od začiatku až po nájdenie hľadaného prvku alebo po prejdenie celého obsahu poľa. Na začiatku sa algoritmus nastaví na prvý prvok poľa a porovná ho s hľadaným prvkom. Ak sa nezhodujú, prejde na ďalší prvok poľa a takto postupuje až po koniec poľa alebo po nájdenie hľadaného prvku. V prípade nájdenia prvku sa algoritmus sekvenčného vyhľadávania končí. Ak sa pri hľadaní dostane až na koniec poľa, znamená to, že v poli hľadaný prvok nie je.

Povedzme, že chceme nájsť prvok s hodnotou 1. Vstupom algoritmu je pole prvkov a hľadaný prvok. Algoritmus postupne porovnáva každý prvok poľa s hľadaným prvkom. V tabuľke 3.1 sú zobrazené ružovou neúspešné porovnania a žltou aktuálne porovnávaný prvok. Počet porovnaní závisí od dĺžky poľa.

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	20	13	15	8	11	1	25	32	19

Tabuľka 3.1: Sekvenčné vyhľadávanie — 3. iterácia

Výstupom algoritmu je súradnica (index) hľadaného prvku v poli, ale len ak bol prvok v poli nájdený. V prípade, že sa v poli prvok nenachádza vráti algoritmus hodnotu, ktorá nemôže byť indexom žiadneho prvku v poli (napríklad -1). V tomto prípade bol hľadaný prvok nájdený na šiestej pozícii (tab. 3.2). [11]

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	20	13	15	8	11	1	25	32	19

Tabuľka 3.2: Sekvenčné vyhľadávanie — 6. iterácia

3.1.2 Sekvenčné vyhľadávanie so zarážkou

Jednou z nevýhod klasického sekvenčného algoritmu bolo to, že pred každým porovnaním, či sa aktuálny prvok rovná hľadanému, muselo ešte urobiť porovnanie, ktoré rozhodlo o tom, či sa nachádza aktuálny index v intervale indexov prehľadávaného poľa. Tento algoritmus umiestni na koniec poľa hľadaný prvok (zarážku) a tak zabezpečí,

aby bol hľadaný prvok vždy nájdený. Ak vo vstupnom poli tento prvok nebol, nájde ho na jeho konci a ukončí sa. Odpadá tak porovnanie, či sa aktuálny index nachádza v rozmedzí indexov poľa.

3.1.3 Vyhľadávanie extrému

Tento algoritmus je určený na nájdenie prvku, ktorý má v celom poli najnižšiu (resp. najvyššiu hodnotu). Pred spustením algoritmu túto hodnotu nepoznáme, ale algoritmus nám po prehľadaní poľa túto hodnotu oznámi.

Vstupom algoritmu je pole, v ktorom chceme zistiť extrémnu hodnotu (maximum alebo minimum). Na začiatku je označený prvý prvok poľa za maximum (resp. minimum). Postupne sa prechádzajú ďalšie prvky poľa a ak sa nájde prvok väčší ako maximum (resp. menší ako minimum), je označený za nové maximum (resp. minimum) poľa. Algoritmus sa zastaví po prehľadaní všetkých prvkov poľa. Výstupom je hodnota extrému alebo jeho poloha v poli.

3.1.4 Vyhľadávanie oboch extrémov súčasne

Vstupom algoritmu je pole, v ktorom chceme zistiť interval, v ktorom sa hodnoty nachádzajú. Na začiatku sa zoberú prvé dva prvky poľa a ten väčší je označený za maximum a menší za minimum. Ďalej algoritmus porovnáva naraz nasledujúce dva prvky v poli. Ten väčší z nich je porovnávaný s maximom a menší s minimom. Ak sa nájde nové maximum alebo minimum, hodnoty sa zaznamenajú a pokračuje sa ďalšou dvojicou prvkov. Pri prehľadávaní poľa môžu nastať dve situácie. Ak má pole párny počet prvkov, tak sa algoritmus jednoducho skončí. Ak má pole nepárny počet prvkov, tak s posledným prvkom sa musí porovnať aj maximum aj minimum. Výstupom algoritmu sú hodnoty maxima a minima. [14]

3.2 Algoritmy pre usporiadané polia

V nasledujúcich algoritmoch budeme predpokladať, že vstupné pole je zoradené vzostupne, teda od najmenšieho prvku po najväčší. Algoritmy môžu fungovať samozrejme aj pre opačné zoradenie, museli by sme však k tomu prispôsobiť rozhodnutia urobené na základe porovnania hľadaného prvku s aktuálnym.

3.2.1 Sekvenčné vyhľadávanie s podmienkou

Podobne ako pri obyčajnom sekvenčnom vyhľadávaní algoritmus prehľadáva prvky poľa od jeho začiatku. V tomto prípade sa však algoritmus nekončí iba v prípade, že prejde celé pole alebo nájde hľadaný prvok, ale algoritmus sa ukončí aj v prípade, že aktuálny prvok je väčší ako hľadaný.

Povedzme, že chceme nájsť prvok s hodnotou 13. Vstupom algoritmu je zoradené pole prvkov, v ktorom budeme vyhľadávať, a prvok, o ktorom chceme vedieť, či sa v poli nachádza. V každom cykle algoritmus zisťuje, či je aktuálny prvok menší ako hľadaný.

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	2	3	5	8	11	12	15	20	21

Tabuľka 3.3: Sekvenčné vyhľadávanie s podmienkou — 5. iterácia

Ak nie je, potom je aktuálny prvok ešte raz porovnaný s hľadaným prvkom. Ak sa tieto prvky rovnajú, hľadaný prvok sa v poli nachádza a algoritmus sa úspešne ukončí. Ak sa prvky nerovnajú, znamená to, že hľadaný prvok sa v poli nenachádza a vyhľadávanie končí. Keďže pole je zoradené a aktuálny prvok je väčší ako hľadaný, je ďalšie prehľadávanie už zbytočné. [6]

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	2	3	5	8	11	12	15	20	21

Tabuľka 3.4: Sekvenčné vyhľadávanie s podmienkou — 7. iterácia

3.2.2 Binárne vyhľadávanie

V usporiadanom poli sa prakticky nepoužíva sekvenčné vyhľadavanie kvôli jeho veľkej časovej náročnosti. Z časových dôvodov je veľmi nevýhodné porovnávať postupne všetky prvky v poli až po nájdenie toho, ktorý hľadáme alebo zistenie, že sa prvok v poli nenachádza. Výhodou zotriedeného poľa je, že môžeme použiť pri hľadaní hodnoty metódu polenia intervalu. Na tejto metóde je založený algoritmus binárneho vyhľadávania. S každou iteráciou sa prehľadávaný interval zmenší na polovicu.

Povedzme, že hľadáme napríklad číslo 20. Vstupom algoritmu je zoradené pole prvkov, v ktorom budeme vyhľadávať, a prvok, o ktorom chceme vedieť, či sa v poli nachádza a ak áno, tak na akej pozícii. V cykle sa vyberie stred poľa (tab. 3.5) a podľa toho, či je hľadaný prvok väčší alebo menší ako zvolený stred, sa vyradí jedna alebo druhá polovica poľa. V tomto prípade sa vyradí ľavá polovica poľa (ružová časť v tab. 3.5) vrátane stredu.

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	2	3	5	8	11	12	15	20	21

Tabuľka 3.5: Binárne vyhľadávanie — 1. iterácia

Ak je hľadaným prvokom práve vybraný stred, tak sa algoritmus úspešne ukončí. V prípade, že stred nie je hľadaným prvkom, tak sa cyklus opakuje až dovtedy, kým stred nebude hľadaným prvkom alebo sa skončí, ak sa prvok v poli nenachádza. Pri úspešnom konci algoritmus vráti pozíciu prvku v poli, pri neúspešnom vráti hodnotu ktorá nemôže byť indexom žiadneho prvku. V tomto prípade algoritmus skončí s výstupom 8 (v tab. 3.6 zelený prvok), pretože na tej pozícii sa našiel hľadaný prvok. [7, 9, 13]

Index prvku	6	7	8	9
Hodnota prvku	12	15	20	21

Tabuľka 3.6: Binárne vyhľadávanie — 2. iterácia

3.2.3 Interpolačné vyhľadávanie

V reálnom živote sa nepoužíva binárny spôsob vyhľadávania, ale skôr sa interpoluje — napr. slovník sa snažíme otvoriť približne v mieste, kde predpokladáme výskyt hľadaného slova. Predpokladom tohto spôsobu vyhľadávania je rovnomerné rozloženie údajov v prehľadávanej dátovej štruktúre. Ak údaje nie sú rovnomerne rozložené je výhodnejšie použiť binárne vyhľadávanie.

Povedzme, že hľadáme napríklad číslo 12. Vstupom algoritmu je zoradené pole prvkov, v ktorom budeme vyhľadávať, a prvok, o ktorom chceme vedieť, či sa v poli nachádza. Na začiatku sa vypočíta podľa vzorca predpokladaná pozícia hľadaného prvku (tab. 3.7 žltý prvok). Ak sa hľadaný prvok nenachádza na vypočítanej pozícii, tak sa postupuje ako pri binárnom vyhľadávaní — vyradí sa jedna alebo druhá časť poľa vrátane vypočítanej pozície (ružová časť v tab 3.7).

Index prvku	1	2	3	4	5	6	7	8	9
Hodnota prvku	2	3	5	8	11	12	15	20	21

Tabuľka 3.7: Interpolačné vyhľadávanie — 1. iterácia

Cyklus sa opakuje až pokým sa na vypočítanej pozícii nenachádza hľadaný prvok prípadne sa hľadanie skončí, ak sa zistí, že sa prvok v poli už nemôže nachádzať. Výstupom je index hľadaného prvku alebo v prípade neúspechu číslo, ktoré nemôže byť indexom žiadneho prvku v poli. V tomto prípade bol prvok nájdený na šiestej pozícii (tab. 3.8 zelený prvok). [10, 15]

Index prvku	6	7	8	9
Hodnota prvku	12	15	20	21

Tabuľka 3.8: Interpolačné vyhľadávanie — 2. iterácia

4 Implementácia a porovnanie algoritmov

Algoritmy som porovnal matematicky podľa priemerného počtu porovnaní a pokusne na náhodne vygenerovanom poli. V poli sa môžu náhodne generované prvky opakovať, keďže som pre jednoduchosť neriešil unikátnosť prvkov. Algoritmy som napísal tak, že sa skončia pri prvej zhode a ďalší výskyt hľadaného prvku už nie je riešený. Všetky algoritmy sú naprogramované ako funkcie do programu MATLAB.

4.1 Pokusná metóda porovnávania algoritmov

Na porovnanie jednotlivých algoritmov som vytvoril program v MATLABe, ktorý každý algoritmus otestoval stokrát na náhodne generovaných poliach. Bolo vygenerované jedno pole, na ktorom boli postupne otestované všetky algoritmy v danej kategórii. Časy hľadaní boli zapisované do tabuliek pre každý algoritmus zvlášť a boli odlišné časy, kedy bol prvok nájdený a kedy nájdený nebol. Ak bol hľadaný prvok nájdený, čas sa do tabuľky zapísal normálne, ak hľadaný prvok nájdený nebol, zapísal sa do tabuľky záporný čas. Časy boli následne rozdelené do dvoch ďalších tabuliek — jedna pre úspešné hľadanie a ďalšia pre neúspešné. V každom riadku boli časy hľadania z jedného rozsahu poľa — napríklad v prvom riadku časy pre pole o rozsahu desaťtisíc prvkov. Z tabuliek boli vytvorené grafy závislosti času na počte prvkov v poli. Na os y v grafe boli vynesené hodnoty mediánov jednotlivých riadkov tabuľky a na os x hodnoty od nuly po dvestotisíc. Porovnané sú grafy úspešného a neúspešného hľadania v rámci jedného algoritmu a potom aj medzi ostatnými algoritmami.

Algoritmy boli testované na počítači s procesorom Intel Core 2 Duo P7550 @ 2,26 GHz, 4 GB RAM, 64-bit Windows 7.

4.2 Algoritmy pre neusporiadané polia

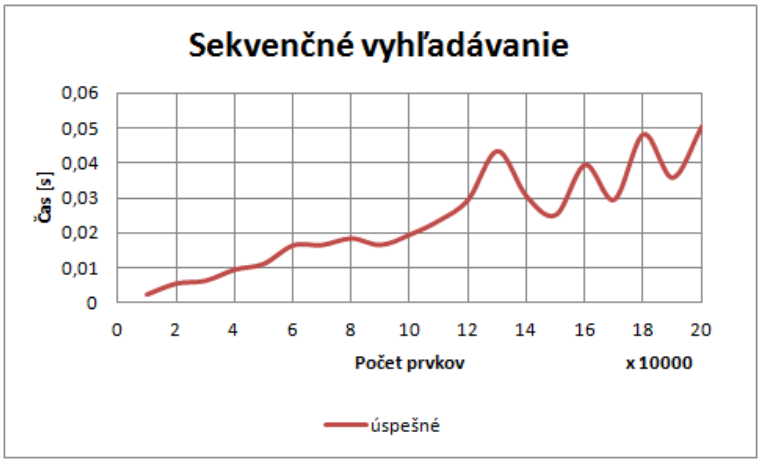
4.2.1 Sekvenčné vyhľadávanie

```
1  function vystup = Sekv_Vyhlad(Pole,Hladany_Prvok)
2    i = 1;
3    pocet_prvkov = length(Pole);
4    vystup = -1;
5    while i <= pocet_prvkov
6      if Pole(i) == Hladany_Prvok
7        vystup = i;
8        break;
9      else
10       i = i +1;
11     end;
12   end;
13 end
```

Princíp činnosti: Do premennej i sa priradí index prvého prvku v poli a do premennej *pocet_prvkov* dĺžka poľa, čo je vlastne počet prvkov, ktoré sa nachádzajú v poli. Potom sa pokračuje cyklom s podmienkou na začiatku, ktorou je, že index aktuálneho prvku poľa musí byť menší alebo rovný ako počet prvkov v poli. Následne sa testuje, či sa aktuálny prvok rovná hľadanému. Ak sa rovnajú, výstupu funkcie sa priradí index prvku, ktorý sa rovnal hľadanému a algoritmus sa ukončí. Ak sa nerovnajú, do premennej i sa priradí číslo o 1 väčšie.

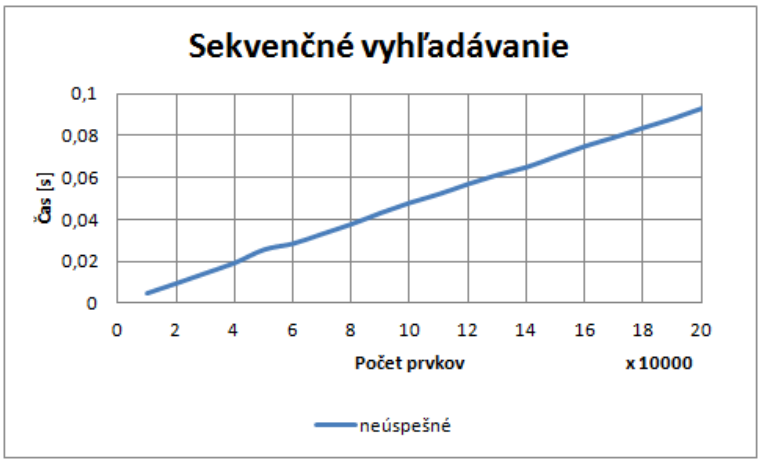
Matematická analýza: Algoritmus porovnáva každý prvok poľa s hľadaným prvkom. V najlepšom prípade, keď sa hľadaný prvok nachádza hneď na začiatku poľa, algoritmus vykoná len 1 porovnanie v cykle a 1 porovnanie aktuálneho prvku s hľadaným, čo v súčte znamená 2 porovnania. V najhoršom prípade, keď sa prvok nachádza na konci poľa, je potreba vykonať n porovnaní v podmienke v cykle, kde n je počet prvkov v poli a n porovnaní aktuálneho prvku s hľadaným. V súčte to je $2n$ porovnaní. Keď sa spriemerujú tieto dve možnosti $(2 + 2n)/2$, tak dostaneme $n + 1$ porovnaní. V prípade neúspešného hľadania sa vždy vykoná $2n + 1$ porovnaní.

Grafická analýza: Na obrázku 4.1 je vidieť, že čas potrebný na nájdenie prvku približne lineárne rastie a veľmi záleží, kde sa nachádzal hľadaný prvok, čo spôsobilo výkyvy v grafe.



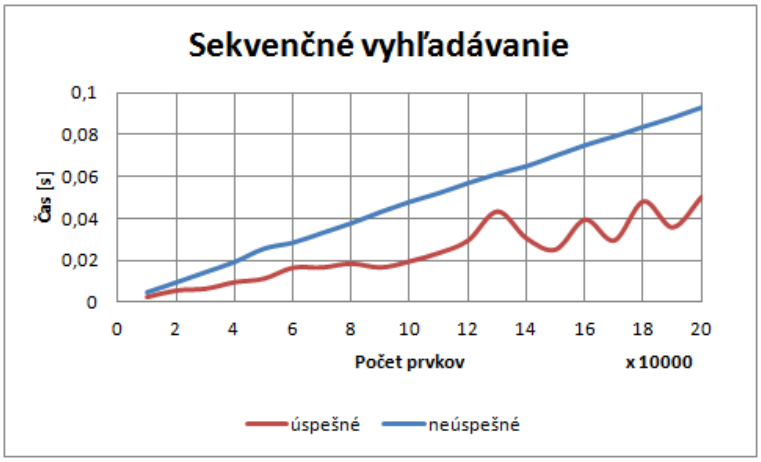
Obr. 4.1: Úspešné sekvenčné vyhľadávanie

Keďže pri neúspešnom hľadaní je prehľadané celé pole, tak čas potrebný na zistenie, že sa prvok v poli nenachádza, rastie lineárne s počtom prvkov, čo znázorňuje graf na obrázku 4.2.



Obr. 4.2: Neúspešné sekvenčné vyhľadávanie

V grafe na obrázku 4.3 vidieť, že čas potrebný na nájdenie prvku je približne polovičný ako pri neúspešnom hľadaní, čo sa zhoduje s odhadmi v matematickej analýze.



Obr. 4.3: Sekvenčné vyhľadávanie

4.2.2 Sekvenčné vyhľadávanie so zarážkou

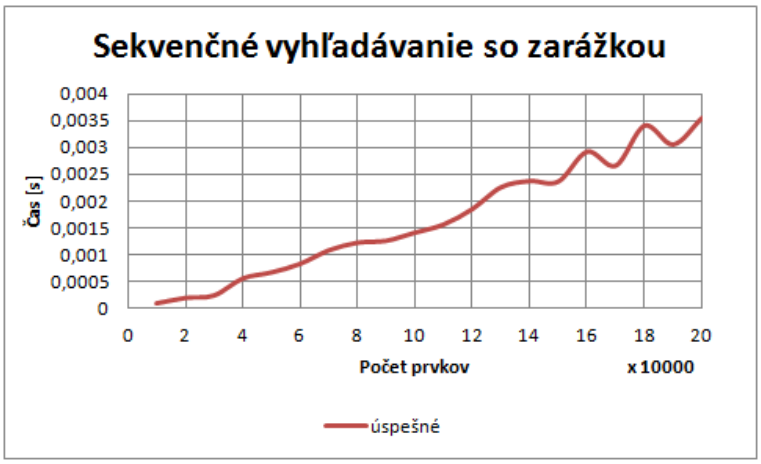
```
1   function vystup = Sekv_Vyhl_Zarazka(Pole, Hladany_Prvok)
2     i = 1;
3     pocet_prvkov = length(Pole);
4     Pole(pocet_prvkov + 1) = Hladany_Prvok;
5     while Pole(i) ~= Hladany_Prvok
6       i = i + 1;
7     end;
8     if i < length(Pole)
9       vystup = i;
10    else
11      vystup = -1;
```

```
12      end;
13    end
```

Princíp činnosti: Do premennej i sa priradí index prvého prvku v poli, do premennej *pocet_prvkov* dĺžka poľa, čo je vlastne počet prvkov, ktoré sa nachádzajú v poli a za posledný prvok poľa sa vloží hľadaný prvok. Potom sa v cykle s podmienkou na začiatku, kde sa porovnáva hľadaný prvok s aktuálnym, zvyšuje hodnota indexu s krokom 1. Cyklus prebieha dovtedy, pokým sa aktuálny prvok nerovná hľadanému. Pomocou zarážky je zaručené, že sa cyklus vždy ukončí úspešne, takže treba ešte testovať, či sa našiel prvok v poli alebo sa hľadanie skončilo na zarážke. Ak je index i menší ako dĺžka poľa, tak do výstupu funkcie sa priradí pozícia nájdeného prvku. V opačnom prípade je výstupom -1, čo znamená, že sa prvok v poli nenachádza.

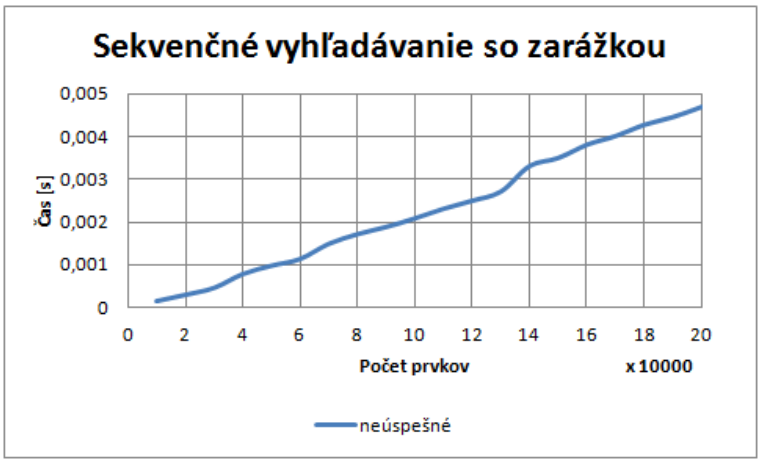
Matematická analýza: Keďže je isté, že prvok sa v poli nachádza, dá sa jednodušiť algoritmus odstránením porovnávania podmienkou *if* v tele cyklu. Oproti obyčajnému sekvenčnému vyhľadávaniu tak odpadne n porovnaní, takže algoritmus vyhodnotí len podmienku na začiatku cyklu $n + 1$-krát. V najlepšom prípade, keď sa hľadaný prvok nachádza hneď na prvej pozícii v poli, sa vykonajú 2 porovnania. Jedno v podmienke cyklu a druhé v teste, či nájdený prvok nie je zarážka. V najhoršom prípade pri nájdení prvku sa vykoná len $n+1$ porovnaní — n porovnaní v podmienke cyklu a 1 porovnanie v teste, či nájdený prvok nie je zarážka. Počet porovnaní pre úspešné vyhľadávanie je po spriemerovaní týchto hodnôt $(n + 3)/2$, čo je približne polovica oproti obyčajnému sekvenčnému vyhľadávaniu. Daňou za zmenšenie počtu porovnaní je réžia spojená s umiestňovaním a prípadným následným odstraňovaním zarážky.

Grafická analýza: Na obrázku 4.4 je vidieť podobný priebeh grafu ako u obyčajného sekvenčného vyhľadávania. Výkyvy krivky sú spôsobené umiestnením hľadaných prvkov v prehľadávanom poli.



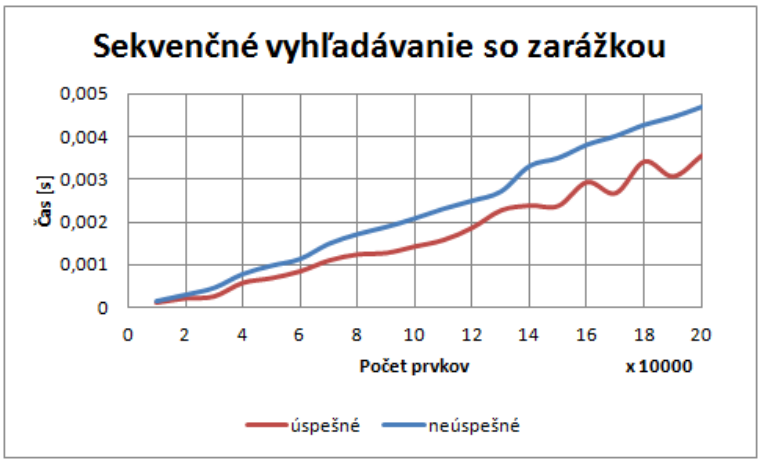
Obr. 4.4: Úspešné sekvenčné vyhľadávanie so zarážkou

Graf na obrázku 4.5 zobrazuje približne lineárnu závislosť času na zistenie, že prvok sa v poli nenachádza, na počte prvkov v poli.



Obr. 4.5: Neúspešné sekvenčné vyhľadávanie so zarážkou

V grafe na obrázku 4.6 je zobrazené úspešné aj neúspešné sekvenčné vyhľadávanie so zarážkou. Čas na nájdenie prvku nie je polovičný voči neúspešnému vyhľadávaniu, ako to bolo u obyčajného sekvenčného vyhľadávania. Je o niečo vyšší kvôli umiestňovaniu zarážky.



Obr. 4.6: Sekvenčné vyhľadávanie so zarážkou

4.2.3 Vyhľadávanie extrému

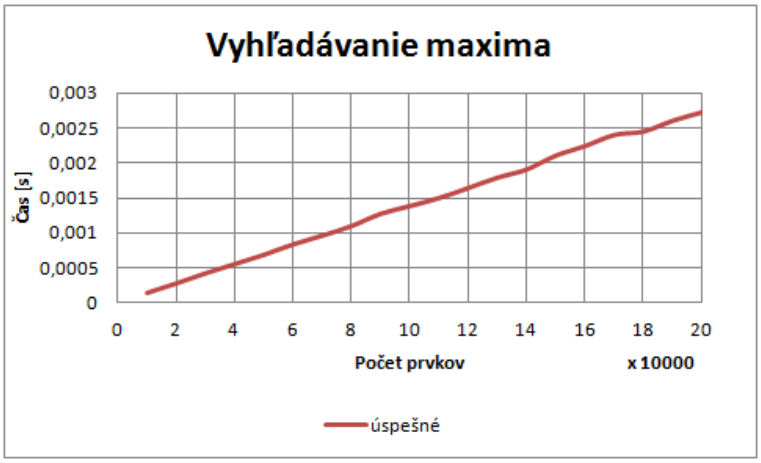
```
1  function vystup = maximum(Pole)
2    i = 2;
3    pocet_prvkov = length(Pole);
4    max = Pole(1);
5    while i <= pocet_prvkov
6      if Pole(i) > max
7        max = Pole(i);
8      end;
9      i = i + 1;
```

```
10      end;
11      vystup = max;
12    end
```

Princíp činnosti: Do premennej i sa priradí index druhého prvku v poli, do premennej max, ktorá reprezentuje maximum, sa priradí hodnota prvého prvku v poli a do premennej $pocet\_prvkov$ dĺžka poľa. Pokračuje sa v cyklom s podmienkou na začiatku, ktorá zaručuje, že cyklus bude prebiehať, pokým bude index prvku menší alebo rovný počtu prvkov v poli. V cykle sa porovnáva aktuálny prvok so súčasným maximom. Ak je aktuálny prvok väčší ako súčasné maximum tak sa priradí do premennej max a stáva sa novým maximom. Na konci sa do výstupu funkcie priradí hodnota premennej max.

Matematická analýza: Algoritmus prehľadáva vždy celé pole a je vždy úspešný. Je potreba vykonať n porovnaní v podmienke cyklu a $n-1$ porovnaní aktuálneho prvku s maximom. Dohromady to teda je $2n-1$ porovnaní.

Grafická analýza: V grafe na obrázku 4.7 je vidiet takmer lineárnu závislosť času potrebného na dokončenie algoritmu na počte prvkov v poli.



Obr. 4.7: Vyhľadávanie maxima

4.2.4 Vyhľadávanie oboch extrémov

```
1   function vystup = max_min(Pole)
2     i = 4;
3     if Pole(1) > Pole(2)
4       max = Pole(1);
5       min = Pole(2);
6     else
7       max = Pole(2);
8       min = Pole(1);
9     end
```

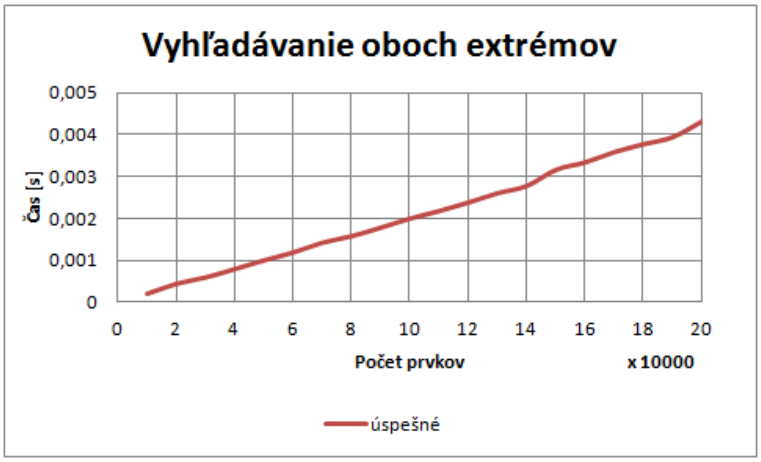
```
10      pocet_prvkov = length(Pole);
11      while i <= pocet_prvkov
12        if Pole(i-1) < Pole(i)
13          if Pole(i-1) < min
14            min = Pole(i-1); end;
15          if Pole(i) > max
16            max = Pole(i); end;
17        else
18          if Pole(i) < min
19            min = Pole(i); end;
20          if Pole(i-1) > max
21            max = Pole(i-1); end;
22        end
23        i = i + 2;
24      end;
25      if i-2 ~= pocet_prvkov
26            if Pole(i-1) > max
27                    max = Pole(i-1); end;
28            if Pole(i-1) < min
29                    min = Pole(i-1); end;
30      end;
31      vystup = [min max];
32    end
```

Princíp činnosti: Do premennej i sa priradí index štvrtého prvku v poli. Potom sa porovnajú prvé dva prvky medzi sebou. Menší prvok sa priradí do premennej *min* reprezentujúcej minimum a väčší do premennej *max* preprezentujúcej maximum. Pokračuje sa cyklom s podmienkou na začiatku, ktorá zaručí, že cyklus bude prebiehať pokým je $i+1$ menšie ako počet prvkov v poli. Prvky sa porovnávajú od tretieho prvku v poli, pretože prvé dva sme už porovnali pred cyklom. V cykle sa zoberú 2 prvky a porovnajú medzi sebou, aby sa zistilo, ktorý je väčší a ktorý menší. Väčší z nich sa porovná s hodnotou v premennej *max*. V prípade, že je väčší ako *max*, je priradený do premennej *max* a stáva sa novým maximom. Menší prvok z dvojice je porovnaný s hodnotou v premennej *min*. Ak je menší ako táto hodnota je priradený do premennej *min* a stáva sa novým minimom.

Matematická analýza: Tento algoritmus taktiež prehľadáva celé pole, takže porovná všetky prvky. Jedno porovnanie sa vykoná pred cyklom aby sa určilo dočasné maximum a minimu z prvých dvoch prvkov. Keďže sa porovnáva až od tretieho prvku tak treba porovnať $n-2$ prvkov. V cykle sa porovnávajú prvky po dvojiciach, takže podmienka v cykle bude testovaná $(n-2)/2+1$-krát. To sa dá upraviť na $n/2$ porovnaní. Dvojica prvkov je medzi sebou porovnaná $(n-2)/2$-krát. Každý prvok je porovnaný buď s minimom alebo s maximom, čo je $n-2$ porovnaní. Ďalej je ešte vykonené porovnanie na konci v prípade poľa s nepárnym počtom prvkov. Ak má pole párny počet prvkov, tak sa vykoná jedno porovnanie, ak nepárny tak sa vykonajú tri porovnania. Dohro-

mady to je pre pole s párnym počtom prvkov $1+(n/2)+(n/2)-1+n-2+1 = 2n-1$ porovnaní. Pre pole s nepárnym počtom prvkov je to $1+(n-1)/2+(n-3)/2+n-3+3 = 2n-1$ porovnaní. Ak sa tieto 2 hodnoty spriemerujú, vyjde $2n-1$ porovnaní. Z toho vyplýva, že je časovo výhodnejšie hľadať oba extrémy naraz, pretože treba urobiť rovnaký počet porovnaní ako pri hľadaní iba jedného extrému.

Grafická analýza: Graf na obrázku 4.8 má, tak isto ako predchádzajúci algoritmus, skoro lineárny priebeh, keďže je vždy potreba prehľadať celé pole.



Obr. 4.8: Vyhľadávanie oboch extrémov naraz

4.3 Algoritmy pre usporiadané polia

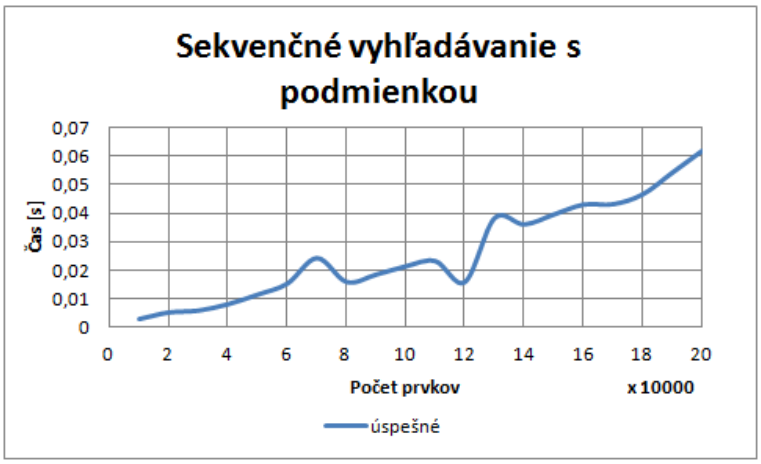
4.3.1 Sekvenčné vyhľadávanie s podmienkou

```
1  function vystup = Sekv_vyhl_podm(Pole, HladanyPrvok)
2    i = 1;
3    pocet_prvkov = length(Pole);
4    vystup = -1;
5    while i <= pocet_prvkov
6      if Pole(i) < HladanyPrvok
7        i= i+1;
8      else
9        if Pole(i) == HladanyPrvok
10         vystup = i;
11         break;
12       else vystup = -1;
13         break;
14       end;
15     end
16   end
17 end
```

Princíp činnosti: Do premennej i sa priradí index prvého prvku v poli, do premennej *pocet_prvkov* dĺžka poľa. Ďalej sa pokračuje do cyklu s podmienkou na začiatku, ktorá zabezpečí, že cyklus bude prebiehať, kým je aktuálny index i menší alebo rovný počtu prvkov v poli. Potom sa porovnáva aktuálny prvok s hľadaným. Ak je aktuálny prvok menší ako hľadaný, tak algoritmus prejde na ďalší prvok v poli. Ak sa aktuálny a hľadaný prvok rovnajú, tak do výstupu funkcie je priradený index prvku i a algoritmus úspešne skončí. V prípade, že aktuálny prvok je väčší ako hľadaný, tak algoritmus skončí neúspešne (výstupom je -1), pretože prvok sa už v poli nemôže nachádzať a ďalšie hľadanie je teda zbytočné.

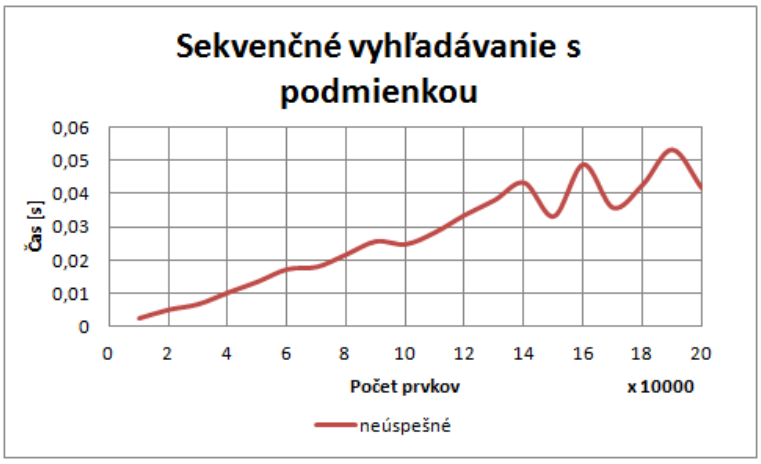
Matematická analýza: Pri úspešnom vyhľadávaní je priemerný počet vykonaných porovnaní podobne ako pri obyčajnom sekvenčnom vyhľadávaní približne n. Výhody tohto algoritmu by sa mali prejaviť hlavne vtedy, keď sa pri vyhľadávaní prvok v poli nenachádza. V takomto prípade totiž algoritmus neprehľadáva prvky až do konca poľa, ale skončí už vtedy, keď je zrejmé, že hľadaný prvok sa už v poli nemôže nachádzať. Ak sa v poli nachádzajú prvky z rovnakého intervalu, z akého prvky vyhľadávame a ich rozdelenie je rovnomerné, počet porovnaní pri neúspešnom vyhľadávaní by mal byť tiež približne n (pri obyčajnom sekvenčnom vyhľadávaní to bolo približne $2n$). Je to preto, lebo ak sa hľadaný prvok nenachádzal na jeho predpokladanom mieste, kde by bol pri úspešnom vyhľadávaní, algoritmus sa ihneď končí. [6]

Grafická analýza: Graf na obrázku 4.9 popisuje závislosť času potrebného na nájdenie hľadaného prvku na počte prvkov v poli. Čas potrebný na nájdenie hľadaného prvku narastá približne lineárne s počtom prvkov v poli. Výkyvy krivky sú spôsobonené umiestnením hľadaného prvku v poli. Ak boli hľadané prvky bližšie ku koncu poľa, tak čas potrebný na ich nájdenie bol vyšší. A naopak, ak boli hľadané prvky bližšie k začiatku poľa, tak čas potrebný na ich nájdenie bol nižší.



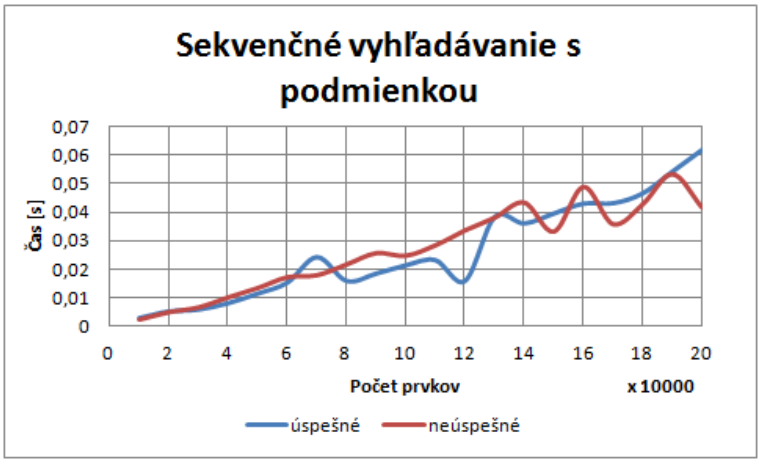
Obr. 4.9: Úspešné sekvenčné vyhľadávanie

Obrázok 4.10 popisuje závislosť času potrebného na zistenie, že sa hľadaný prvok v poli nenachádza, na počte prvkov v poli. Výkyvy sú spôsobené tým, že algoritmus zistil, že hľadaný prvok sa už nemôže ďalej nachádzať v poli a tak sa ukončil.



Obr. 4.10: Neúspešné sekvenčné vyhľadávanie

Obrázok 4.11 ukazuje porovnanie grafov úspešného a neúspešného vyhľadávania. Dokazuje tvrdenie z matematickej analýzy, že čas vyhľadávania môže byť kratší pri neúspešnom vyhľadávaní ako pri úspešnom, keďže vyhľadávanie sa končí v momente, keď hľadaný prvok je menší ako aktuálny a neprehľadáva sa zbytok poľa ako pri klasickom sekvenčnom vyhľadávaní.



Obr. 4.11: Sekvenčné vyhľadávanie s podmienkou

4.3.2 Binárne vyhľadávanie

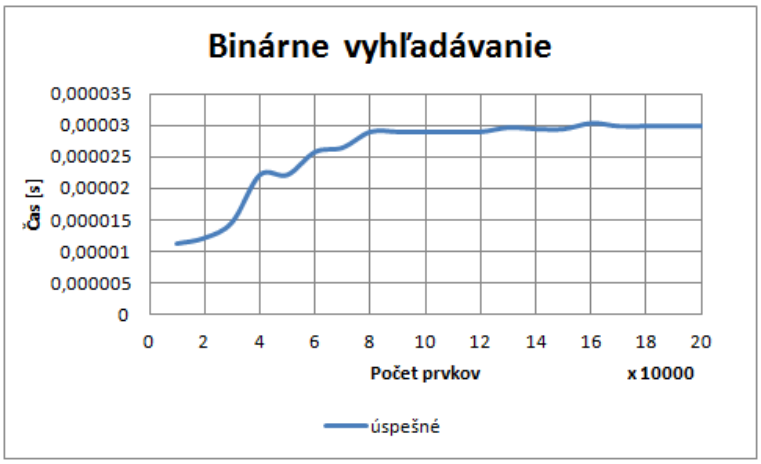
```
1  function vystup = bin(Pole, HladanyPrvok)
2    imin = 1;
3    imax = length(Pole);
4    vystup = -1;
5      while imin <= imax
6        stred = round((imin + imax)/2);
7        if Pole(stred) < HladanyPrvok
8          imin = stred + 1;
```

```
 9          elseif Pole(stred) > HladanyPrvok
10            imax = stred - 1;
11          else vystup = stred;
12            break;
13          end;
14        end;
15    end
```

Priníp činnosti: Do premennej *imin* sa priradí index prvého a do *imax* index posledného prvku poľa. Ďalej sa pokračuje do cyklu s podmienkou na začiatku, ktorá zaručí, že cyklus bude prebiehať pokým je spodná hranica intervalu *imin* menšia alebo rovná hornej hranici intervalu *imax*. Potom sa v cykle vypočíta stred poľa (intervalu) podľa jednoduchého vzorca $(imin - imax)/2$ a zaokrúhli sa na celé číslo. Vypočítaný stred je následne testovaný, či je väčší alebo menší ako hľadaný prvok. Ak je menší, tak sa vyradí pravá strana intervalu, ak je väčší tak ľavá. Ak nie je ani väčší ani menší tak sa našiel hľadaný prvok, výstupu funkcie je priradená hodnota *stred*u a algoritmus sa úspešne skončí.

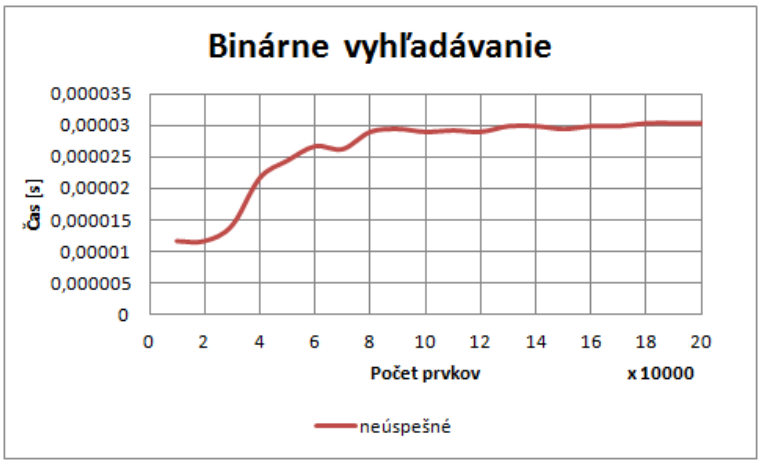
Matematická analýza: Keďže algoritmus pri každom cykle delí pole na polovicu, delí sa vyhľadávací problém vždy na problém polovičný. Počet cyklov je teda v najhoršom prípade približne $log_2 n$. V priemernom prípade úspešného hľadania je hľadaný prvok nájdený v predposlednom kroku, čo znamená, že priemerná časová zložitosť algoritmu je približne $log_2 n - 1$ porovnaní. [7]

Grafická analýza: Graf na obrázku 4.12 popisuje závislosť času potrebného na nájdenie hľadaného prvku na počte prvkov v poli. Čas potrebný na nájdenie hľadaného prvku narastá približne logaritmicky s počtom prvkov v poli, čo potvrdzuje matematickú analýzu. Výkyvy krivky sú spôsobonené umiestnením hľadaného prvku v poli.



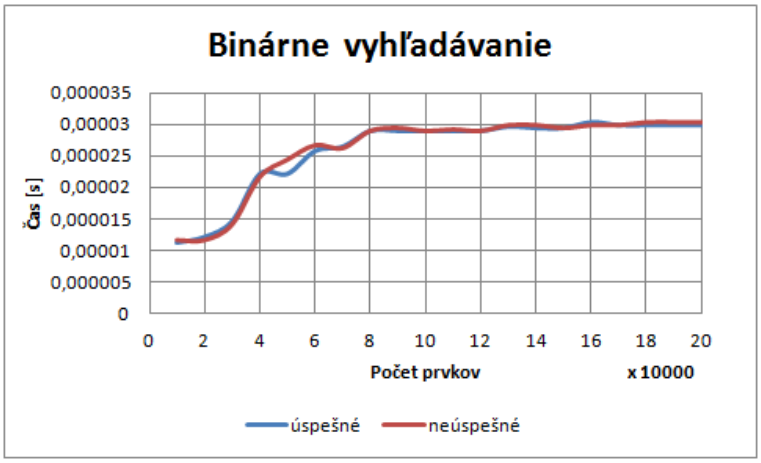
Obr. 4.12: Úspešné binárne vyhľadávanie

Graf na obrázku 4.13 popisuje závislosť času potrebného na zistenie, že sa hľadaný prvok v poli nenachádza, na počte prvkov v poli. Čas potrebný na neúspešné binárne vyhľadávanie narastá približne logaritmicky s počtom prvkov v poli.



Obr. 4.13: Neúspešné binárne vyhľadávanie

Obrázok 4.14 ukazuje porovnanie oboch grafov úspešného a neúspešného vyhľadávania. Z grafu vyplýva, že časy vyhľadávania pri úspešnom a neúspešnom hľadaní sú skoro identické, takže dá sa povedať, že poloha prvku v poli nemá veľký vplyv na rýchlosť. Teda dĺžka vyhľadávania závisí hlavne na počte prvkov v poli.



Obr. 4.14: Binárne vyhľadávanie

4.3.3 Interpolačné vyhľadávanie

```
1  function vystup = interpol(Pole, HladanyP)
2    imin = 1;
3    imax = length(Pole);
4    vystup = -1;
5    while Pole(imin) <= HladanyP && HladanyP <= Pole(imax)
```

```
6       menovatel = Pole(imax) - Pole(imin);
7       if menovatel == 0
8         if Pole(imin) == HladanyP
9           vystup = imin;
10          break;
11        else vystup = -1;
12          break;
13        end
14      end
15      stred = imin+(HladanyP-Pole(imin))*((imax-imin)/menovatel);
16      stred = round(stred);
17      if Pole(stred) == HladanyP
18        vystup = stred;
19        break;
20      elseif Pole(stred) < HladanyP
21        imin = stred + 1;
22      else imax = stred - 1;
23      end
24    end
25  end
```

Priníp činnosti: Do premennej *imin* sa priradí index prvého a do *imax* index posledného prvku poľa. Ďalej sa pokračuje cyklom s podmienkou na začiatku, ktorá zaručí, že cyklus bude prebiehať, ak je hľadaný prvok v intervale poľa. Potom sa vypočíta menovateľ z rozdielu minima a maxima. Ak by bol menovateľ rovný nule, tak sa testuje, či je minimum rovné hľadanému prvku. Ak sa rovná tak výstupu funkcie je priradený index uložený v premennej *imin*, v opačnom prípade sa priradí hodnota -1. Ak sa menovateľ nerovná nule cyklus pokračuje ďalej. Vypočíta sa *stred* intervalu — pravdepodobné miesto výskytu prvku, podľa vzorca:

$$(imin) + (hlp - min) \cdot \frac{imax - imin}{menovatel}$$

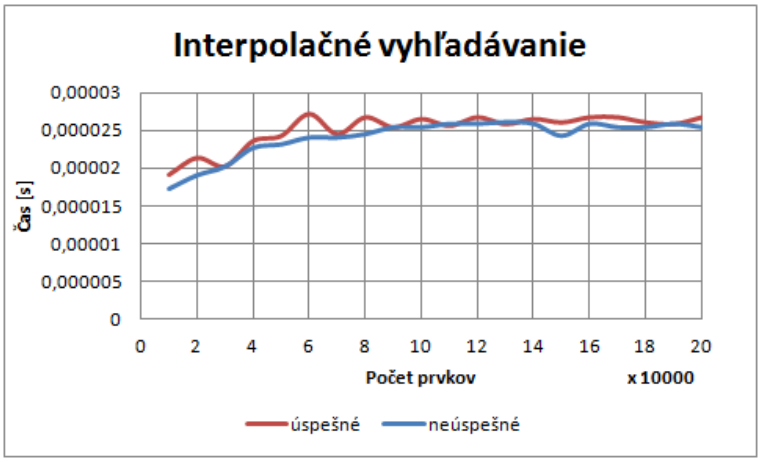
imin — index najmenšieho prvku poľa
imax — index najväčšieho prvku poľa
min — hodnota najmenšieho prvku poľa
hlp — hodnota hľadaného prvku
menovatel — rozdiel hodnôt najväčšieho a najmenšieho prvku poľa

Potom sa testuje, či sa na vypočítanom mieste nachádza hľadaný prvok. Ak tam nie je, postupuje sa rovnako ako pri binárnom vyhľadávaní — podľa toho, či je hľadaný prvok väčší alebo menší ako vypočítaná pozícia tak sa vyradí ľavá alebo pravá strana intervalu.

Matematická analýza: Interpolačné vyhľadávanie vykoná približne *log log n* porovnaní a to aj pri úspešnom aj neúspešnom vyhľadávaní za predpokladu, že prvky

poľa sú nezávisle rovnomerne distribuované náhodné čísla. Môže byť rýchlejšie ako binárne vyhľadávanie pre polia, ktoré majú sto a viac prvkov. [1]

Grafická analýza: Graf úspešného hľadania na obrázku 4.15 sa dá aproximovať funkciou $log\ log\ x$, ktorá veľmi pomaly rastie a teda je minimálny časový rozdiel medzi hľadaním v poliach s vyššími počtami prvkov. Tak isto je vidieť, že čas na úspešné a neúspešné hľadanie je približne rovnaký a má skoro identický priebeh.



Obr. 4.15: Interpolačné vyhľadávanie

4.4 Vyhodnotenie algoritmov

4.4.1 Neusporiadané polia

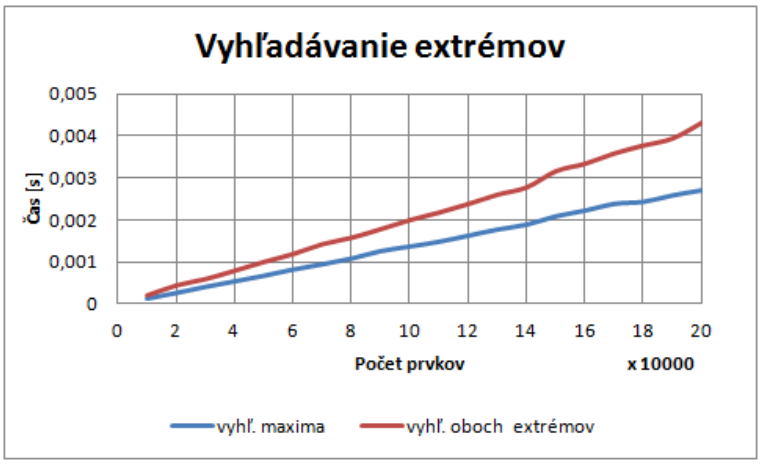
Sekvenčné vyhľadávania: V grafe na obrázku 4.16 sú zobrazené grafy úspešného sekvenčného vyhľadávania bez zarážky a so zarážkou. Z grafu jasne vidieť, že sekvenčné vyhľadávanie so zarážkou je omnoho rýchlejšie ako obyčajné sekvenčné vyhľadávanie.

Obr. 4.16: Porovnanie algoritmov sekvenčného vyhľadávania

Počet prvkov	Čas $sekv1$ $[ms]$	Čas $sekv2$ $[ms]$
10000	2,4322	0,09578
50000	11,1961	0,6739
100000	19,4862	1,4184
150000	25,1359	2,3736
200000	50,3561	3,5608

Tabuľka 4.1: Tabuľka časov úspešných sekvenčných vyhľadávaní

Vyhľadávanie extrémov: V grafe na obrázku 4.17 je zobrazený graf vyhľadávania jedného extrému a graf vyhľadávania oboch extrémov súčasne. Z grafu aj z tabuľky 4.2 vyplýva, že vyhľadávanie jedného extrému je približne o tretinu rýchlejšie ako vyhľadávanie oboch extrémov súčasne. Aj keď je vyhľadávanie jedného extrému rýchlejšie, stále je lepšie použiť vyhľadávanie oboch extrémov súčasne.



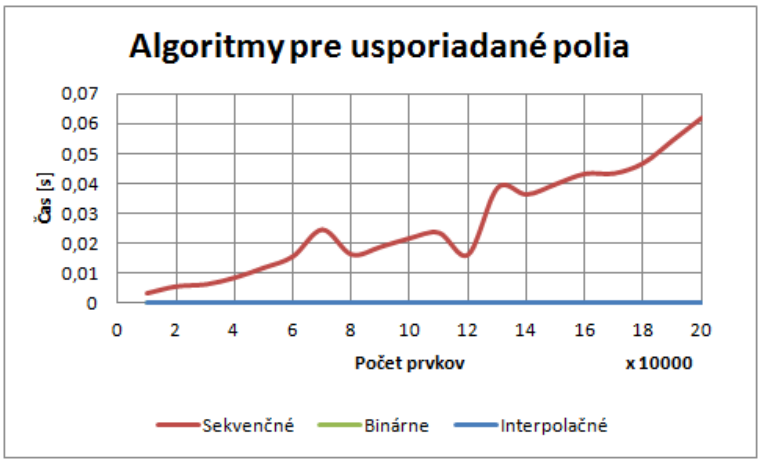
Obr. 4.17: Porovnanie algoritmov pre vyhľadávanie maxima

Počet prvkov	Čas max $[ms]$	Čas $maxmin$ $[ms]$
10000	0,1422	0,2133
50000	0,6857	1,0086
100000	1,3795	2,0054
150000	2,0991	3,1652
200000	2,7146	4,3156

Tabuľka 4.2: Tabuľka časov vyhľadávaní extrémov

4.4.2 Usporiadané polia

V grafe na obrázku 4.18 sú zobrazené grafy sekvenčného, binárneho a interpolačného vyhľadávania. Je jasne vidieť, že sekvenčné vyhľadávanie trvá neporovnateľne dlhšie ako binárne či interpolačné. Preto v ďalšom porovnávaní už sekvenčné vyhľadávanie nebude uvažované.

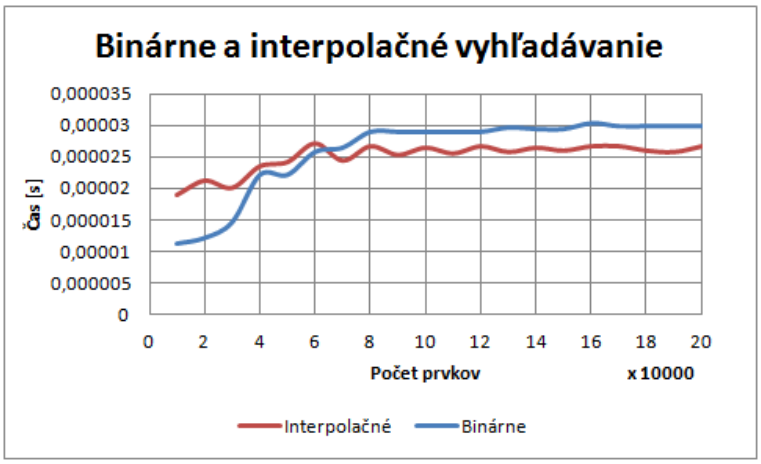


Obr. 4.18: Porovnanie algoritmov pre usporiadané polia

Počet prvkov	Čas $sekv3$ $[ms]$	Čas bin $[ms]$	Čas int $[ms]$
10000	3,1923	0,01132	0,01902
50000	11,7608	0,02219	0,02423
100000	21,6531	0,02898	0,02649
150000	39,7727	0,02944	0,02604
200000	61,9833	0,02989	0,02672

Tabuľka 4.3: Tabuľka časov sekvenčného, binárneho a interpolačného vyhľadávania

V ďalšom grafe na obrázku 4.19 sú zobrazené už len časy úspešného binárneho a interpolačného vyhľadávania. Binárne vyhľadávanie dosahuje lepších časov do približne šesťdesiattisícprvkového poľa. Pri viacprvkovom poli čas potrebný na nájdenie prvku u binárneho vyhľadávania narastá, ale u interpolačného ostáva približne rovnaký. To súvisí s priebehom funkcie $log\ log\ n$, ktorá veľmi pomaly rastie, a funkciou $log\ n$.



Obr. 4.19: Porovnanie binárneho a interpolačného vyhľadávania

Z výsledkov vyplýva, že najrýchlejší a teda aj najlepší algoritmus vyhľadávania pre neusporiadané pole je sekvenčné vyhľadávanie so zarážkou. Pre usporiadané pole to nie je také jednoznačné. Veľmi záleží na počte prvkov v poli. Ak máme pole prvkov do šesťdesiat tisíc tak výhodnejšie je použiť binárne vyhľadávanie. Pre väčšie polia vyšlo lepšie interpolačné vyhľadávanie, ktoré dosahovalo približne rovnaké časy hľadaní pri viac ako stotisícprvkovom poli a lepšie časy ako binárne už pri sedemdesiattisícovom poli.

5 Záver

V praxi dnes existujú rôzne metódy ukladania a vyhľadávania informácií. Táto práca sa zameriava hlavne na vyhľadávanie v jednorozmerných poliach, ktoré si môžme predstaviť ako riadok tabuľky.

Cieľom práce bolo porovnanie rýchlosti, s akou algoritmy dokážu nájsť hľadaný prvok v poli alebo zistiť, že sa prvok v poli nenachádza. Porovnanie jednotlivých algoritmov bolo realizované matematicky na základe priemerného počtu porovnaní a pokusne na náhodne vygenerovaných poliach. Podrobné výsledky porovnania sú uvedené v kapitole Implementácia a porovnanie algoritmov. Tieto výsledky potvrdili, že na neusporiadané jednorozmerné pole je najvhodnejšie použiť sekvenčné vyhľadávanie so zarážkou. Na usporiadané jednorozmerné pole je najvhodnejšie použiť binárne vyhľadávanie ak pole je menšie ako šesťdesiattisícprvkové a na polia s väčším počtom prvkov je vhodnejšie použiť interpolačnú metódu. Interpolačnú metódu je však vhodné použiť iba vtedy ak je známe, že dáta sú rovnomerne rozložené. V opačnom prípade je vhodnejšia binárna metóda.

Literatúra

[1] WIEDERMANN, J. *Vyhledávání*. Praha, SNTL 1991. 142 s. ISBN 80-03-00537.

[2] WIRTH, N. *Algoritmy a štruktúry údajov*. Bratislava, ALFA 1988.

[3] HONZÍK, J. a kol. *Programovací techniky. Skripta*. VUT v Brně, 1985. 357 s.

[4] ŠOLTIS, Martin. *Vyhľadávanie* [online]. Bratislava, 2004 [cit. 2013-03-13]. Dostupné z: http://www2.fiit.stuba.sk/~pospichal/soltis/uvod.htm. Záverečný projekt. STU.

[5] ŠOLTIS, Martin. *Vyhľadávanie v binárnych stromoch* [online]. Bratislava, 2004 [cit. 2013-03-13]. Dostupné z: http://www2.fiit.stuba.sk/~pospichal/soltis/kapitola3.htm. Záverečný projekt. STU.

[6] ŠOLTIS, Martin. *Algoritmy pre usporiadané polia* [online]. Bratislava, 2004 [cit. 2013-03-13]. Dostupné z: http://www2.fiit.stuba.sk/~pospichal/soltis/algo121.htm. Záverečný projekt. STU.

[7] ŠOLTIS, Martin.*Binárne vyhľadávanie* [online]. Bratislava, 2004 [cit. 2013-03-13]. Dostupné z: http://www2.fiit.stuba.sk/~pospichal/soltis/algo122.htm. Záverečný projekt. STU.

[8] WEISSTEIN, Eric W. *Binary Tree* [online]. [cit. 2013-03-23]. Dostupné z: http://mathworld.wolfram.com/BinaryTree.html

[9] MIČKA, Pavel. *Binární vyhledávání* [online]. [cit. 2013-04-29]. Dostupné z: http://www.algoritmy.net/article/21/Binarni-vyhledavani

[10] MIČKA, Pavel. *Interpolační vyhledávání* [online]. [cit. 2013-04-29]. Dostupné z: http://www.algoritmy.net/article/160/Interpolacni-vyhledavani

[11] MIČKA, Pavel. *Lineární vyhledávání* [online]. [cit. 2013-04-29]. Dostupné z: http://www.algoritmy.net/article/19/Linearni-vyhledavani

[12] VEČERKA, Arnošt. *Základní algoritmy* [online]. Olomouc, 2007 [cit. 2013-04-29]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/zakladni_alg.pdf

[13] *Binární vyhledávání* [online]. [cit. 2013-04-29]. Dostupné z: http://www.devbook.cz/algoritmus-binarni-vyhledavani-v-setridenem-poli

[14] *Hledání extrému (minima a maxima)* v poli [online]. [cit. 2013-04-29]. Dostupné z: http://www.devbook.cz/algoritmus-hledani-extremu-minima-a-maxima-v-poli

[15] *Interpolační vyhledávání* [online]. [cit. 2013-04-29]. Dostupné z: http://www.devbook.cz/algoritmus-interpolacni-vyhledavani-v-setridenem-poli

[16] *Binární vyhledávací strom* [online]. [cit. 2013-04-29]. Dostupné z: http://www.devbook.cz/algoritmus-vyhledavaci-struktury-binarni-vyhledavaci-strom-bst

[17] *Binary Search Trees* [online]. [cit. 2013-04-29]. Dostupné z: http://epaperpress.com/sortsearch/bin.html

[18] *Hash Tables* [online]. [cit. 2013-04-29]. Dostupné z: http://epaperpress.com/sortsearch/has.html